MACSio Development and Proxy Application Validation

Elsa Gonsiorowski, James Dickson, Mark Miller
Lawrence Livermore National Laboratory, Livermore, CA, USA

DOE COE Performance Portability Meeting
Denver, CO August 22–24, 2017

Abstract

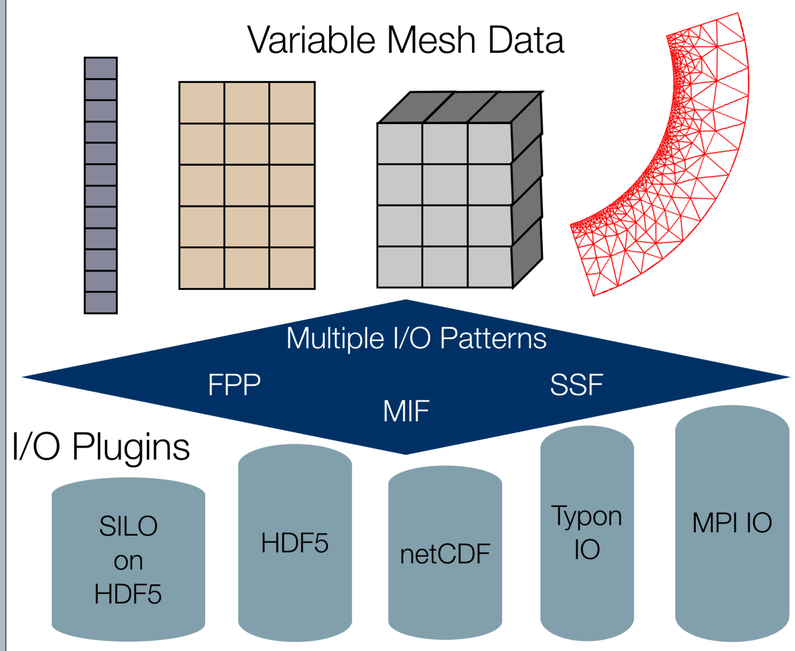
Proxy applications are a vital resource for evaluating and preparing for new systems and hardware. A proxy application for I/O workloads will be essential as new storage tiers and vendor solutions are being developed and deployed.

MACSio is a multi-purpose, application-centric, scalable I/O proxy application designed to imitate a variety of multi-physics applications. Using a plug-in structure, it operates at the same level of abstraction as real applications. That is, MACSio can utilize a wide selection of the I/O software stack. It also implements a variety of multi-physics code features to closely mimic the way in which data flows in-to and out-of these applications.

This poster describes the architecture of the MACSio proxy application and the various development efforts in the project throughout the last year. These development efforts include new features for manipulating dataset meshes and basic compute workload implementations.

In addition to feature developments, an effort is underway to validate and quantify MACSio's approximation to a given multi-physics application. This effort includes the development of an I/O tracing tool, a tool for analyzing the resultant traces, and a formula to quantify the similarity between two such traces.

MACSio Architecture



Related Work

Open Source on GitHub

MACSio is currently under active development. It is available at https://github.com/llnl/macsio. Several development efforts are underway and we welcome contributions from others in the community.

Current efforts include implementing a number of software engineering best practices. These include several efforts to increase the ease of building MACSio and software libraries required for the various plugins. Work is underway to convert the build system to CMake and to create a Spack package. MACSio currently supports Doxygen documentation, but additional resources (such as a getting started guide) are being developed.

Publication

Dickson, James, Wright, Steven A., Maheswaran, Satheesh, Herdman, J. A., Harris, Duncan, Miller, Mark C. and Jarvis, Stephen A. (2017) *Enabling portable I/O analysis of commercially sensitive HPC applications through workload replication*. In: Cray User Group 2017, Redmond, California, USA, 7-12 May 2017.Published in: Cray User Group 2017 Proceedings (CUG2017 Proceedings) pp. 1-14.

This paper uses Darshan I/O characterization and the MACSio proxy application to replicate five production workloads, showing how these can be used effectively to investigate I/O performance when migrating between HPC systems ranging from small local clusters to leadership scale machines. Preliminary results indicate that it is possible to generate datasets that match the target application with a good degree of accuracy. This enables a predictive performance analysis study of a representative workload to be conducted on five different systems. The results of this analysis are used to identify how workloads exhibit different I/O footprints on a file system and what effect file system configuration can have on performance.

Lawrence Livermore National Laboratory

MACSio Architecture

Available Plugins

A key design feature of MACSio is the use of a dynamic, run-time plugin architecture. This design defines how high-level data generated within MACSio is delivered to I/O plugins and will enable plugin developers the greatest flexibility in deciding how best to handle data marshaled by MACSio. MACSio currently implements plugins for a number of I/O software libraries. Thus, MACSio is able to operate at the same level-of-abstraction as the applications it approximates.

Currently MACSio includes plugins for SILO (LLNL), HDF5, TyponIO (UK Mini-App Consortium), and MPI IO. Development of an ADIOS plugin is underway.

Data Generation

MACSio generates data that is marshaled for I/O performance testing. The generated data is parameterized, distributed 1D, 2D, or 3D meshes (structured or unstructured) and arrays. Parameterizations include:

- Nominal and optionally randomization of mesh/array part counts per rank/core
- A few choices in how mesh/array parts are distributed in parallel including nicely structured distributions as well as fully unstructured distributions and distributions that utilize only a subset of all cores/ranks.
- Nominal and optionally randomization of mesh/array part sizes (in terms of numbers of zones of mesh or array entries)
- The number, type (char, int, float) and kind (nodal/zonal) of variables (mesh or arrays)
- Choice in algorithm used to fill variable buffers with data (e.g., constant, random/chaotic, sinusoidal, Poison distributed, etc.)
- Depth and breadth with optional randomization of auxiliary metadata hierarchies (such as might be seen in material models or various other rich metadata produced by multi-physics codes)
- Frequency of main mesh/array dump writes, reads or both as well as being optionally interleaved with auxiliary data dumps.

The data MACSio generates will be housed in a JSON-like object tree that is handed off to plugins for dump writes and received from plugins on dump reads. This tree will include information essential for plugins to determine parallel distribution of the main data objects.

I/O Patterns

SSF: Single Shared File

In the SSF paradigm, parallelism is achieved through concurrent access to a single, shared file (from the perspective of the application). This paradigm is sometimes also called N->1 because it is N tasks writing to one, single file. In this paradigm I/O requests can be either independent or collective. However, collective requests are seen as being somewhat unique to the SSF paradigm as well as potentially offering the greatest opportunity for high performance. On the other hand, there are some subtleties regarding what *collective* I/O operations truly mean in the SSF. SSF requires a parallel interface at the filesystem level (e.g. Lustre, GPFS, or PLFS).

In some descriptions, the SSF paradigm is further divided into *segmented* and *strided* access patterns. These access patterns have to do with the *granularity* at which data from different tasks intermingles in the file address space. In segmented SSF, large swaths of the file address space tend to be read/written by a single task. In strided SSF, data from many/all tasks tends to co-mingle even in very fine grained swaths.

MIF: Multiple Independent File

In the MIF paradigm, parallelism is achieved through simultaneous access to multiple files. The application divides itself into file groups. For each file group, the application manages exclusive access among all the tasks of the group. I/O is *serial* within groups but parallel *across* groups. The number of files (groups) is wholly independent from the number of processors and is often chosen to match the number of independent I/O pathways available in the hardware between the compute nodes and the filesystem. This paradigm is sometimes also called N->M because it is N tasks writing to M files (M<N).

In this paradigm I/O requests are almost exclusively independent. However, there are scenarios where collective I/O requests can be made to work and might even make sense in the MIF paradigm. The onus is on the application to manage the distribution of data across potentially many files. In truth, this illuminates the only salient distinction between SSF and MIF. In either paradigm, if you dig deep enough into the I/O stack, you soon discover that data is always being distributed across multiple files. The only difference is whether that physical arrangement of data is hidden from the application by some sort of higher level abstraction (e.g. a parallel filesystem) or explicitly managed by the application (and thereby also exposed to the filesystem).

FPP: File Per Process

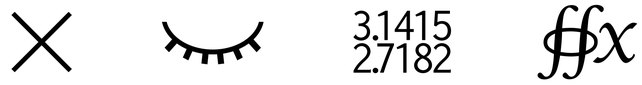
The file per processor paradigm is just a special case of MIF where the number of files is equal to the number of processors. This paradigm is sometimes called N->N because it is N tasks writing to N files. FPP paradigms typically also include a throttle to govern the number of files that are being accessed at any one time to avoid overloading the underlying storage systems components.

New Features

The past year has seen several new features implemented within MACSio. As always, new features are added with two contradictory goals in mind. The objective of increasing flexibility is sometimes at odds with maintaining simplicity and general application emulation.

Improvements to Mesh Structure

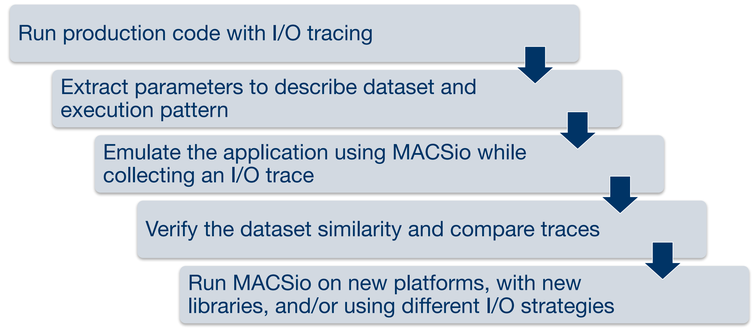
The data within real physics applications is dynamic. Both during initial problem set-up and throughout execution. Current development efforts are underway to improve MACSio's mesh capabilities. First, there will be support for multiple meshes and mesh types within a single run. Second, there will support for dataset evolution during a single run. This emulates mesh-refinement applications, where the overall size of the dataset grows throughout execution. In addition, work is being do to support addition data formats. MACSio will soon support generic JSON data, similar to that generated by other LLNL software tools



Compute Workload

MACSio now has 4 options for emulating a compute workload. First, as before, there is the option for no compute work done between I/O phases. Next, there is a basic *sleep* option, where MACSio is idle for a given time period. Third, is the option for each process to compute random floating point operations. Finally, there is the option for MACSio to perform a parallel solve operation, similar to ones in compute benchmarks.

Validation Process



Understanding the ways in MACSio is similar or different to a give multi-physics application requires quantifying the approximation. This includes evaluating the similarities between the resulting dataset (both in terms of size and number of files) as well as evaluating the I/O operations which occur during execution. Through this understanding, application developers will be able to relate the performance of MACSio on new hardware or using a different I/O software stack to the target application. The performance of new I/O approaches can be evaluated quickly and without change to origin application.

Initial Results

An initial implementation of a tracer was used to profile an example problem of a real physics application. A MACSio run was then designed to emulate the example problem. This run was also traced. A tally of the SILO library function calls can be seen in the table at right.

As a basic comparison, MACSio was able to create the same number of files, with an accuracy of 99.5% in terms of bytes written to file. Work remains to be done to:

- Create a tool for analyzing traces
- Create a standard for evaluating the similarity between two traces

Function	MACSio	Application
DBOpenReal	9	9
DBPutUcdMesh	9	9
DBPutZonelist2	9	9
DBPutUcdvar	414	395
DBPutUcdvar1	414	395
DBClose	9	9
DBWrite	0	302

Conclusion

With new computer architectures becoming available, the need for MACSio is apparent. While many development efforts are focused on GPUs, MACSio can be used to understand how I/O performance is affected. This requires no domain knowledge from the multi-physics application community.

New MACSio features improve the quality of workload emulation, both in terms of data I/O as well as compute operations. These features improve dataset emulation and increase accuracy throughout an execution.

It is essential that MACSio performance is validated and quantified for a given multi-physics application. With a validated MACSio, we hope that it will be seen as tool for evaluating new hardware, software, and I/O strategies. It can then be used to guide application development and improve performance of I/O operations.

National Nuclear Security Administration

LLNL-POST- 737158

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, and partially funded by the Laboratory Directed Research and Development Program at LLNL under tracking code 12-ERD-005.